Geometric Modeling

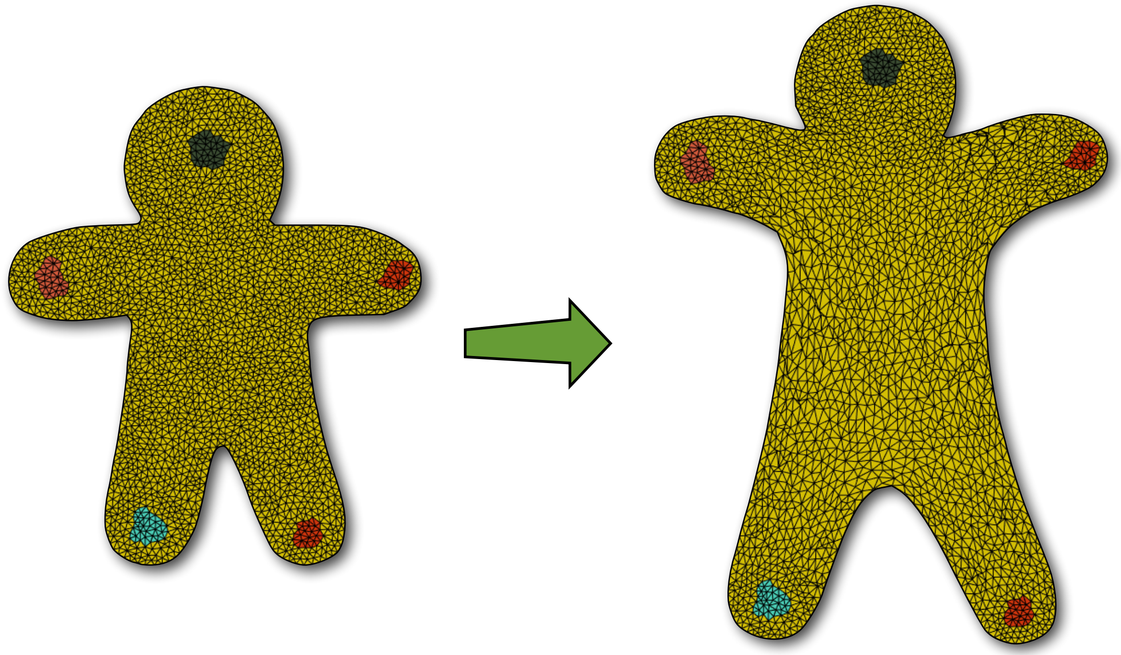
Assignment 5: Shape Deformation

Julian Panetta — fjp234@nyu.edu

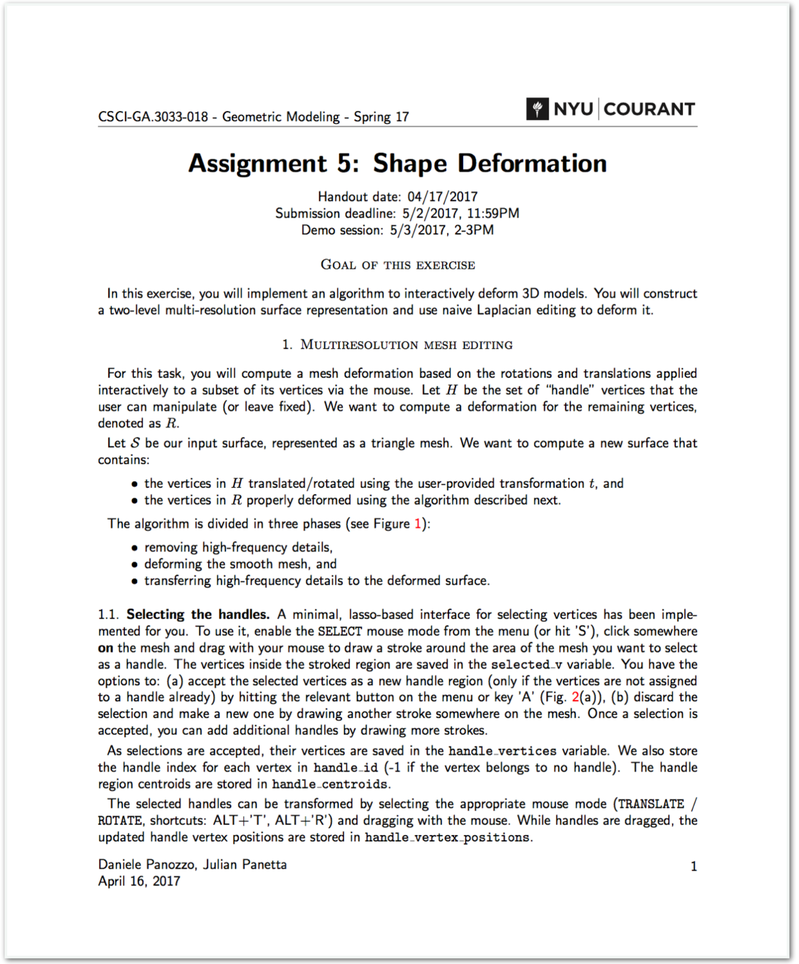
Acknowledgements: Olga Diamanti

NYU | COURANT

Shape Deformation



- **Deadline: 5/2/2017 11:59PM**



CSCI-GA.3033-018 - Geometric Modeling - Spring 17

NYU | COURANT

Assignment 5: Shape Deformation

Handout date: 04/17/2017
Submission deadline: 5/2/2017, 11:59PM
Demo session: 5/3/2017, 2-3PM

Goal of this exercise

In this exercise, you will implement an algorithm to interactively deform 3D models. You will construct a two-level multi-resolution surface representation and use naive Laplacian editing to deform it.

1. Multiresolution mesh editing

For this task, you will compute a mesh deformation based on the rotations and translations applied interactively to a subset of its vertices via the mouse. Let H be the set of "handle" vertices that the user can manipulate (or leave fixed). We want to compute a deformation for the remaining vertices, denoted as R.

Let $\mathcal{S}$ be our input surface, represented as a triangle mesh. We want to compute a new surface that contains:

- the vertices in H translated/rotated using the user-provided transformation t, and
- the vertices in R properly deformed using the algorithm described next.

The algorithm is divided in three phases (see Figure 1):

- removing high-frequency details,
- deforming the smooth mesh, and
- transferring high-frequency details to the deformed surface.

1.1. **Selecting the handles.** A minimal, lasso-based interface for selecting vertices has been implemented for you. To use it, enable the SELECT mouse mode from the menu (or hit 'S'), click somewhere **on** the mesh and drag with your mouse to draw a stroke around the area of the mesh you want to select as a handle. The vertices inside the stroked region are saved in the `selected_v` variable. You have the options to: (a) accept the selected vertices as a new handle region (only if the vertices are not assigned to a handle already) by hitting the relevant button on the menu or key 'A' (Fig. 2(a)), (b) discard the selection and make a new one by drawing another stroke somewhere on the mesh. Once a selection is accepted, you can add additional handles by drawing more strokes.

As selections are accepted, their vertices are saved in the `handle_vertices` variable. We also store the handle index for each vertex in `handle_id` (-1 if the vertex belongs to no handle). The handle region centroids are stored in `handle_centroids`.

The selected handles can be transformed by selecting the appropriate mouse mode (`TRANSLATE` / `ROTATE`, shortcuts: ALT+'T', ALT+'R') and dragging with the mouse. While handles are dragged, the updated handle vertex positions are stored in `handle_vertex_positions`.

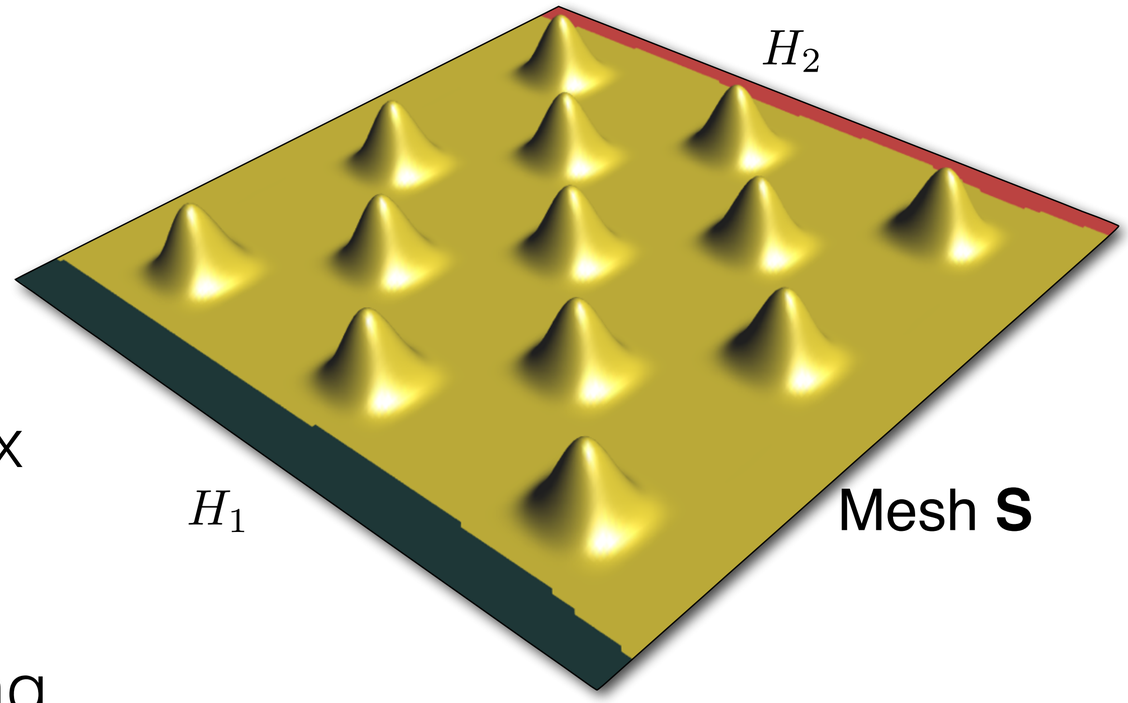
Daniele Panozzo, Julian Panetta
April 16, 2017

1

NYU | COURANT

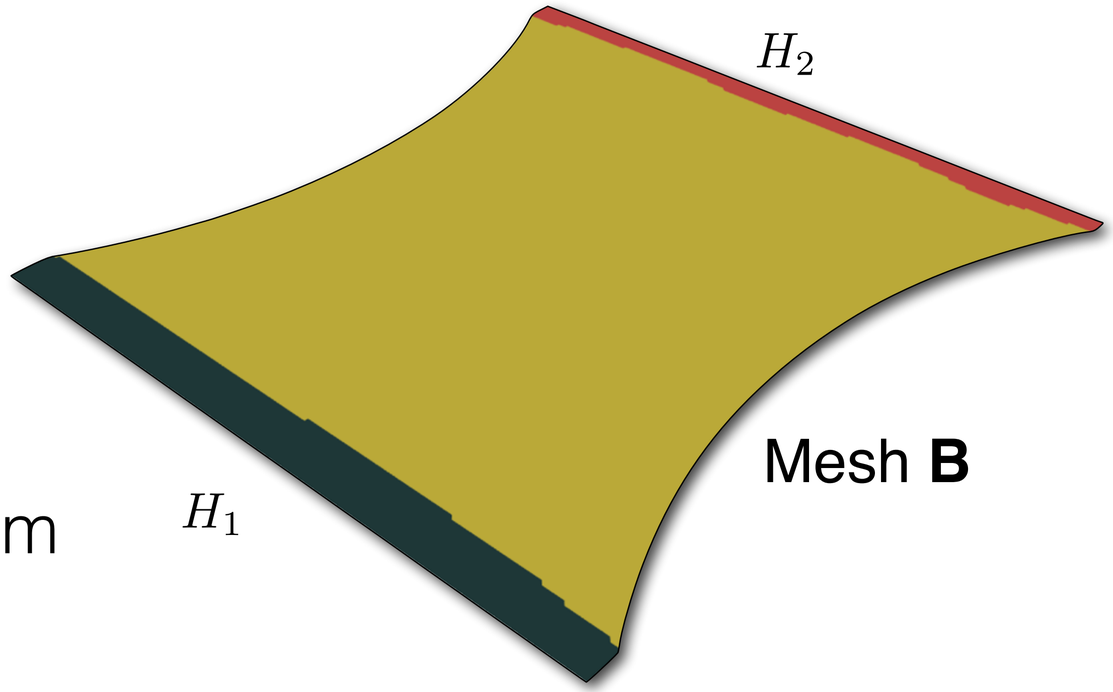
Step 1: Select and Deform Handle Regions

- Draw vertex selection with mouse
- Move one handle H at a time to the deform mesh
- Leave some handles undeformed to fix vertices.
- Code provided for the picking/dragging



Step 2: Smooth Mesh

- Remove high-frequency details from unconstrained vertices.
- This smooth mesh will be deformed; details are added back afterward
- Smooth by solving a bi-laplacian system (minimize the Laplacian Energy)



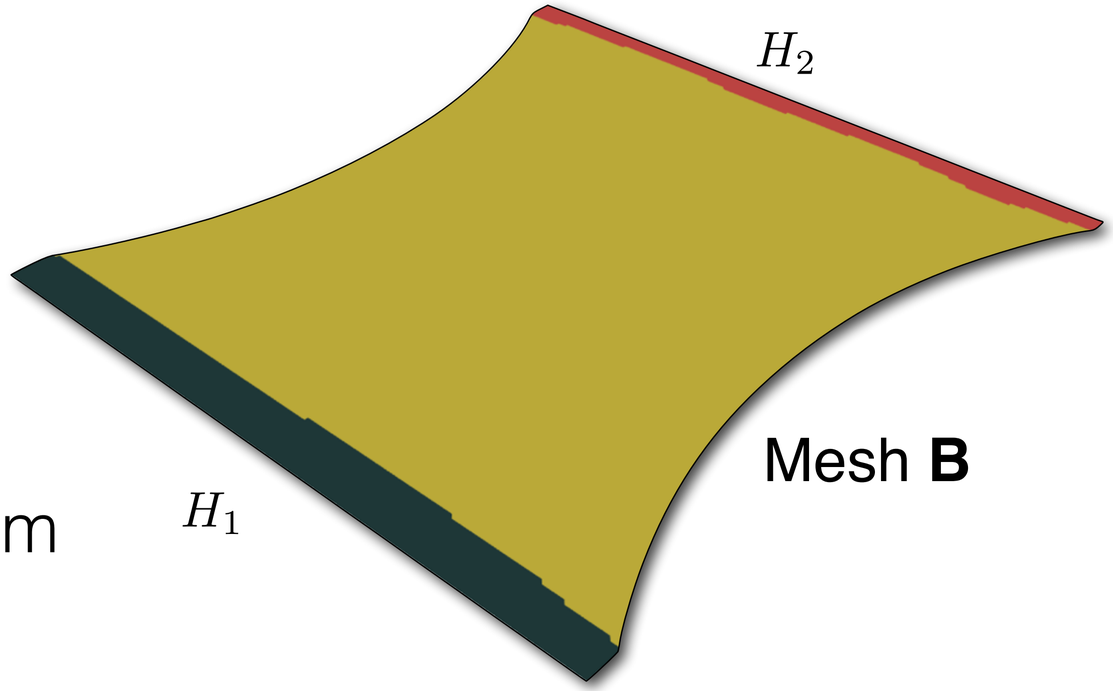
Step 2: Smooth Mesh

- Remove high-frequency details from unconstrained vertices.
- This smooth mesh will be deformed; details are added back afterward
- Smooth by solving a bi-laplacian system (minimize the Laplacian Energy):

$$\min_{\mathbf{v}} \mathbf{v}^T \mathbf{L}_\omega \mathbf{M}^{-1} \mathbf{L}_\omega \mathbf{v}$$

$$\text{s.t.} \quad \mathbf{v}_{H_i} = o_{H_i} \ \forall i$$

Original vertex positions of handles



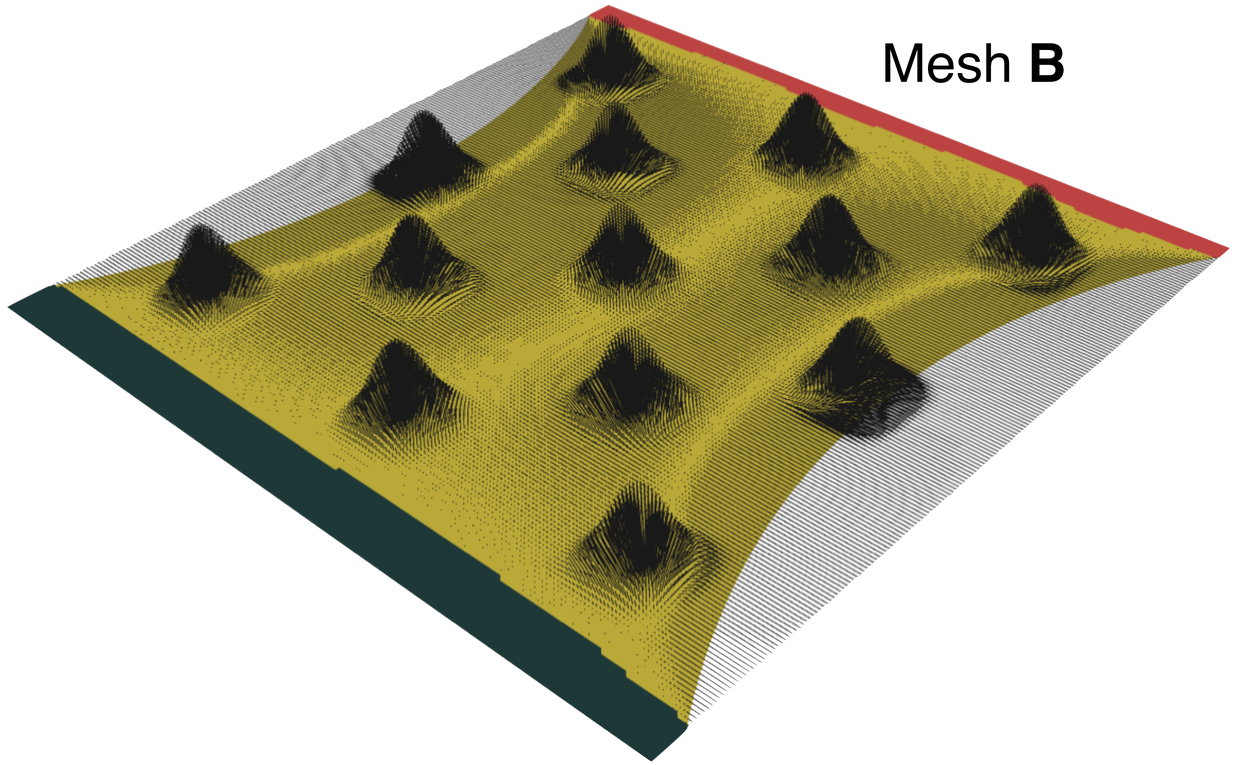
Step 3: Encode Displacements

- Compute the per-vertex displacements from B to S:

$$\mathbf{d}_i = \mathbf{v}_i^S - \mathbf{v}_i^B$$

- These represent the **details** of the original surface.

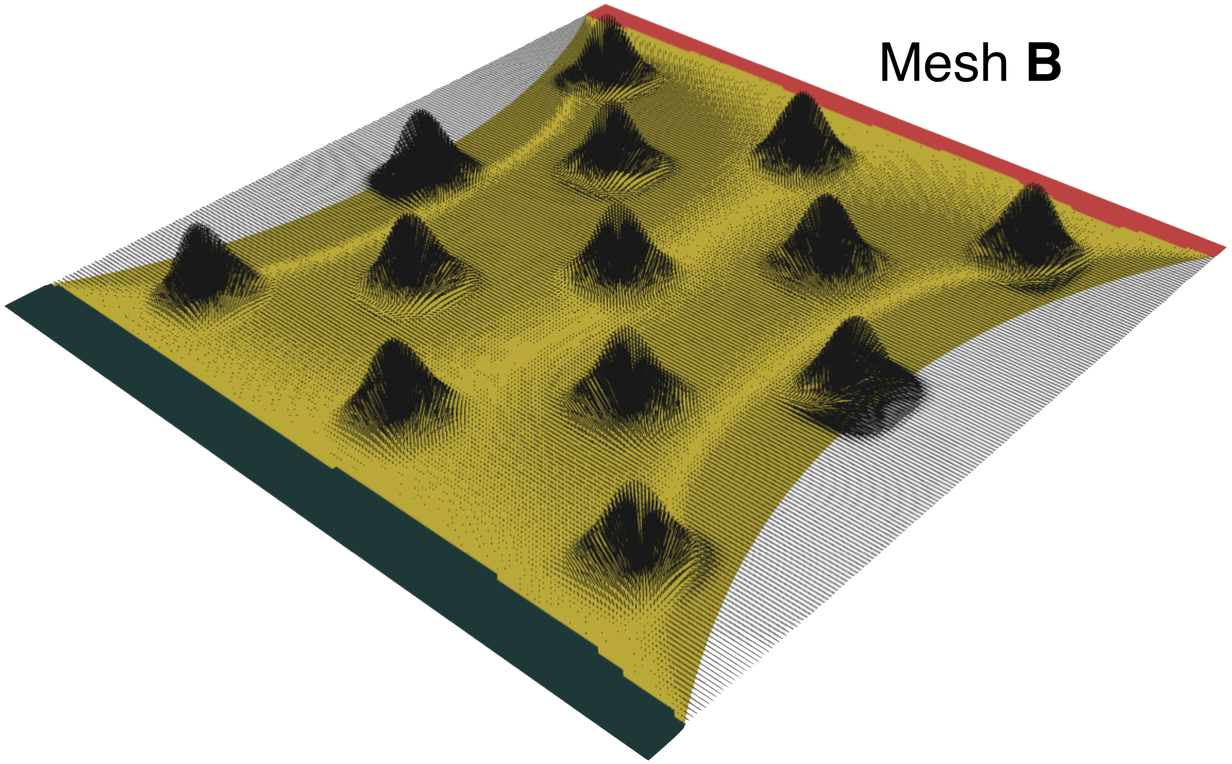
- We will use these to add back (transformed) details after the deformation



Step 3: Encode Displacements

$$\mathbf{d}_i = \mathbf{v}_i^S - \mathbf{v}_i^B$$

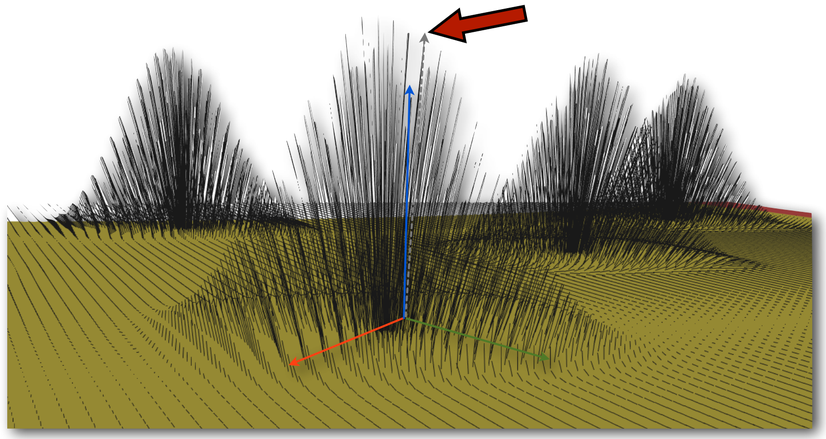
- We want these details to rotate with Mesh **B** as it is later deformed into Mesh **B'**
- To do this, we express the displacements in a **local frame on B,** which is then rotated to align with **B'**
- We just need to define the basis vectors for this frame…

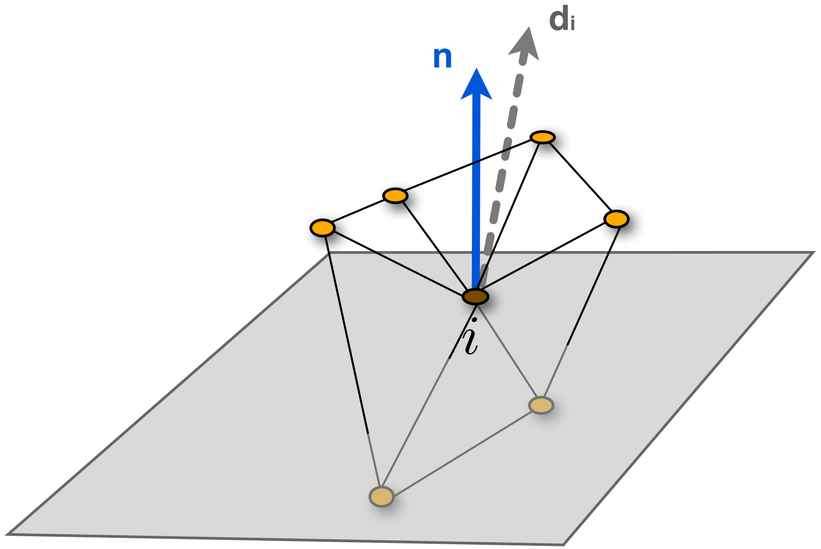


Step 3: Encode Displacements

- Construct the local frame for vertex i:

  1. Calculate normal $\mathbf{n_i}$ (for surface **B**)



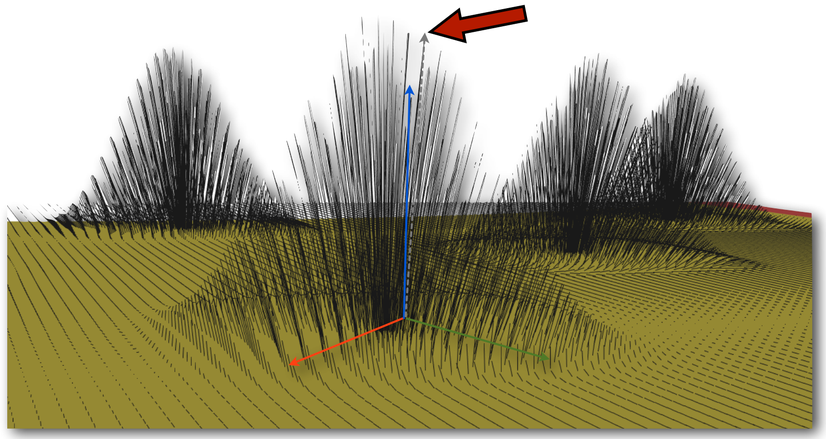


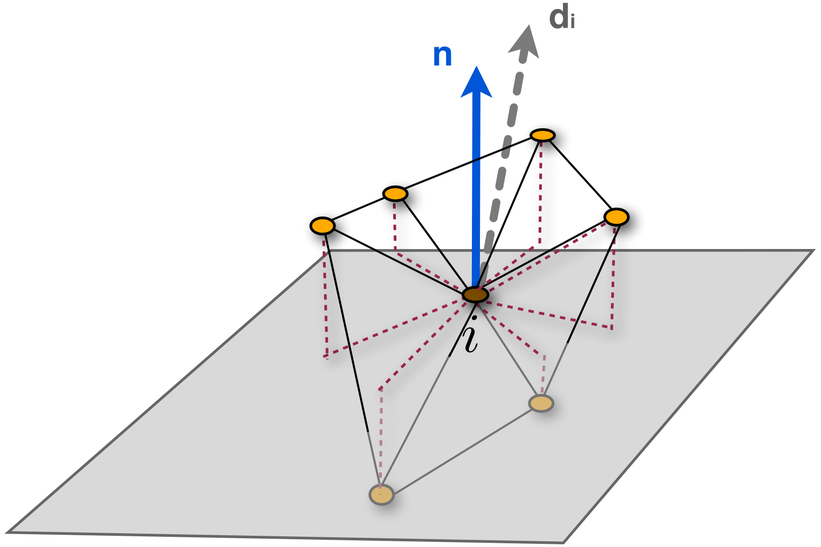
Step 3: Encode Displacements

- Construct the local frame for vertex i:

  1. Calculate normal $\mathbf{n_i}$ (for surface **B**)

  2. Project all neighboring vertices to the tangent plane (perpendicular to $\mathbf{n_i}$)





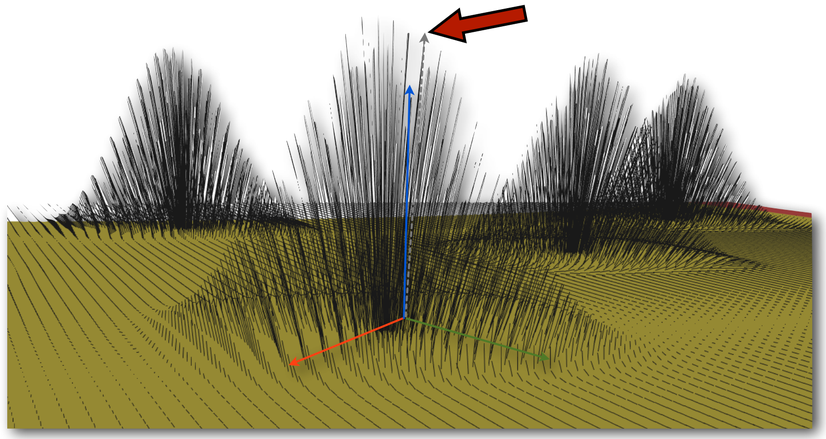
Step 3: Encode Displacements

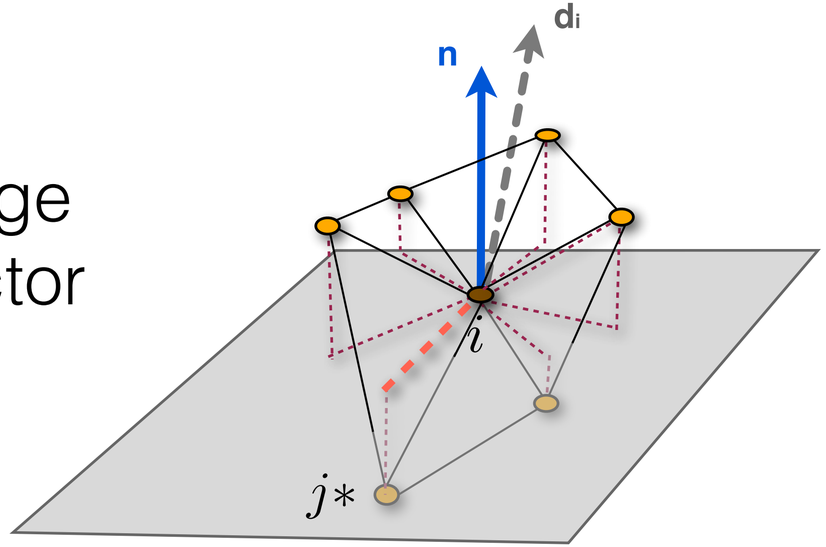
- Construct the local frame for vertex i:

  1. Calculate normal $\mathbf{n_i}$ (for surface **B**)

  2. Project all neighboring vertices to the tangent plane (perpendicular to $\mathbf{n_i}$)

  3. Find neighbor j* for which projected edge (i, j) is longest. Normalize this edge vector and call it $\mathbf{x_i}$.





Step 3: Encode Displacements

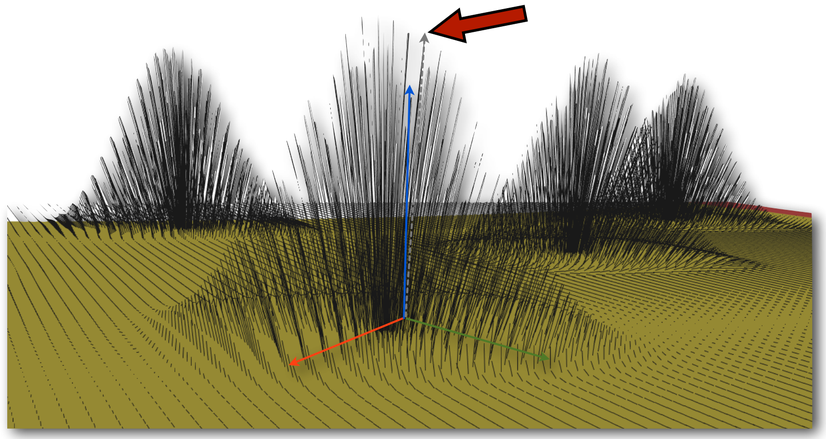
- Construct the local frame for vertex i:

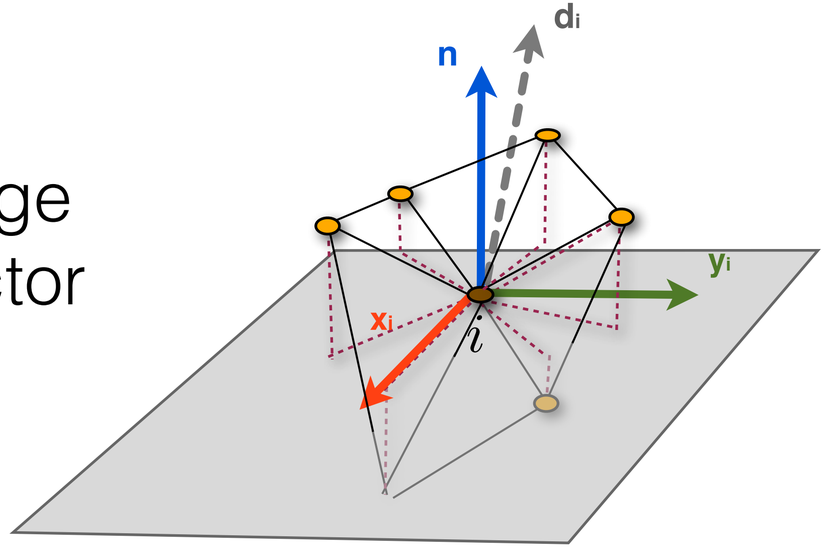
  1. Calculate normal $\mathbf{n_i}$ (for surface **B**)

  2. Project all neighboring vertices to the tangent plane (perpendicular to $\mathbf{n_i}$)

  3. Find neighbor j* for which projected edge (i, j) is longest. Normalize this edge vector and call it $\mathbf{x_i}$.

  4. Construct $\mathbf{y_i}$ using the cross product, completing orthonormal frame ($\mathbf{x_i}$, $\mathbf{y_i}$, $\mathbf{n_i}$)



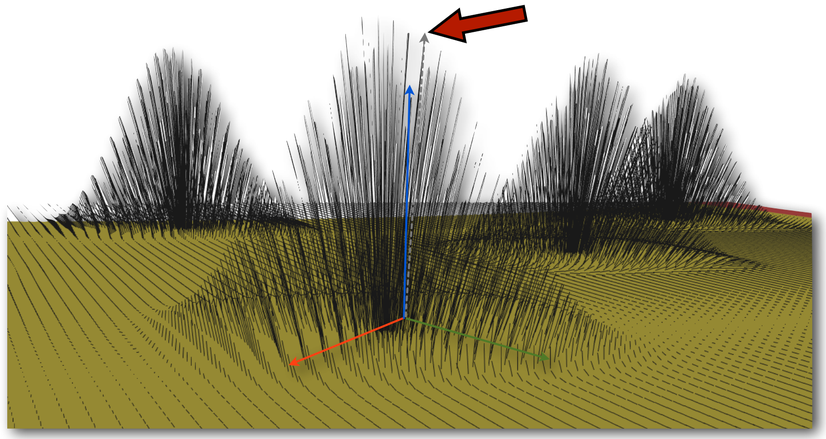


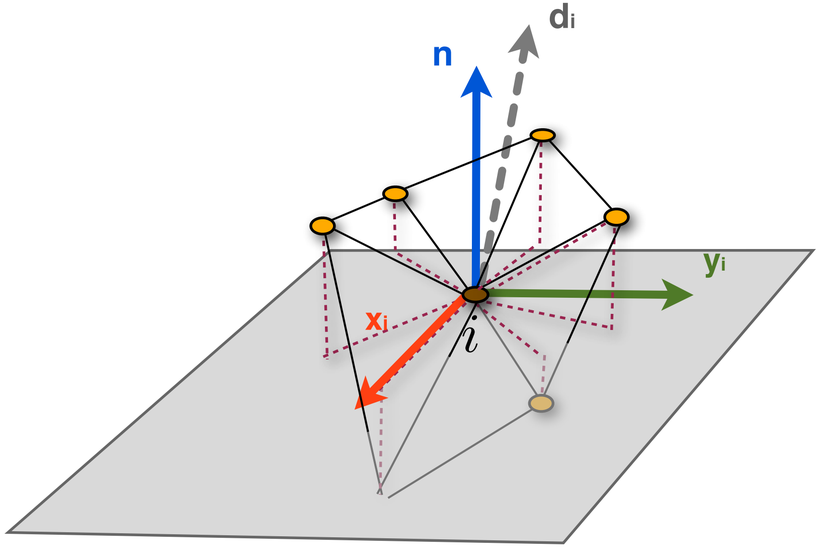
Step 3: Encode Displacements

- Decompose the displacement vectors in the frame's basis:

$$\mathbf{d}_i = d_i^{\mathbf{x}}\mathbf{x}_i + d_i^{\mathbf{y}}\mathbf{y}_i + d_i^{\mathbf{n}}\mathbf{n}_i$$

- (The basis is orthonormal, so you can do this just with inner products.)





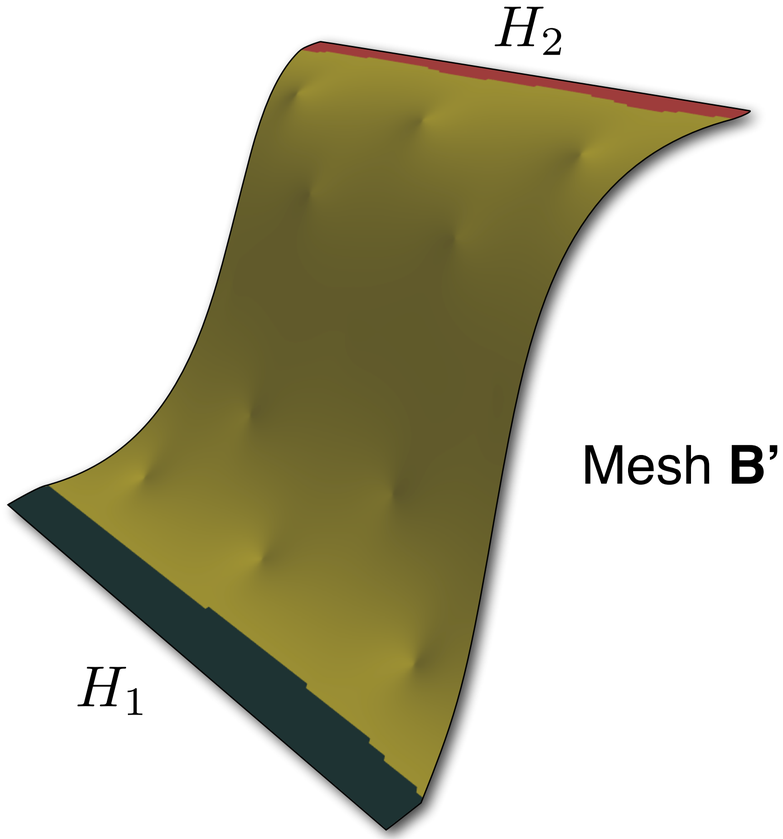
Step 4: Deform **B**

- User manipulates the handles
- You solve for the deformed smooth mesh, **B’**
- Solve bi-laplacian system again, using the new handle position as constraints:

$$\min_{\mathbf{v}} \mathbf{v}^T \mathbf{L}_\omega \mathbf{M}^{-1} \mathbf{L}_\omega \mathbf{v}$$

$$\text{s.t.} \quad \mathbf{v}_{H_i} = t(o_{H_i}) \; \forall i$$

Transformed vertex positions at handles.

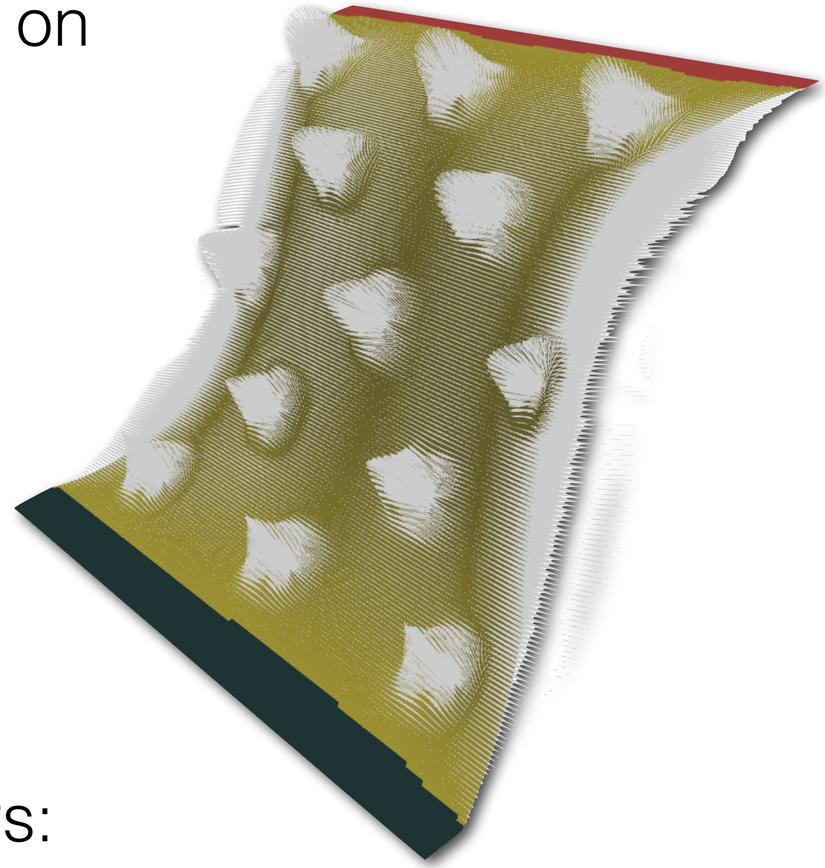


Mesh **B’**

Step 5: Add Transformed Detail

- Compute for each vertex v_i the new local frame on **B'**
  - Calculate normal $\mathbf{n}_i'$ (for surface **B'**)
  - Compute new frame basis ($\mathbf{x}_i'$, $\mathbf{y}_i'$, $\mathbf{n}_i'$) as before, but using the **same edge** (i, j*) as chosen for **B** (so frames are compatible).
- Construct the transformed displacement vectors:

$$\mathbf{d}_i' = d_i^{\mathbf{x}}\mathbf{x}_i' + d_i^{\mathbf{y}}\mathbf{y}_i' + d_i^{\mathbf{n}}\mathbf{n}_i'$$

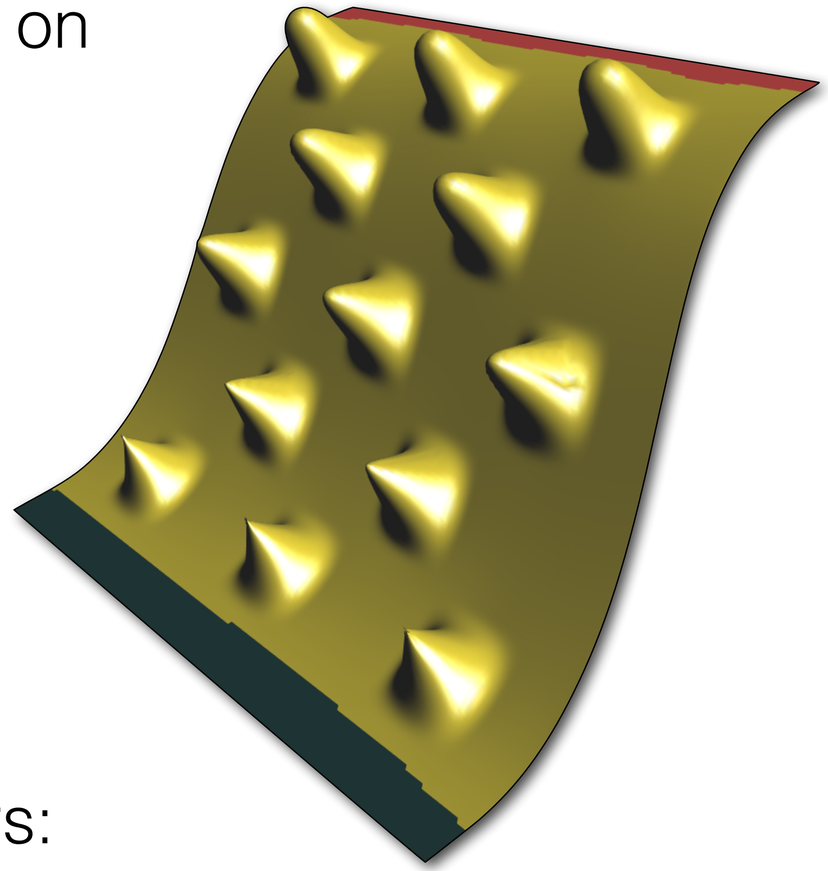


Step 5: Add Transformed Detail

- Compute for each vertex v_i the new local frame on **B'**
  - Calculate normal $\mathbf{n}_i'$ (for surface **B'**)
  - Compute new frame basis ($\mathbf{x}_i'$, $\mathbf{y}_i'$, $\mathbf{n}_i'$) as before, but using the **same edge** (i, j*) as chosen for **B** (so frames are compatible).
- Construct the transformed displacement vectors:

$$\mathbf{d}_i' = d_i^{\mathbf{x}} \mathbf{x}_i' + d_i^{\mathbf{y}} \mathbf{y}_i' + d_i^{\mathbf{n}} \mathbf{n}_i'$$

- Add add them to **B'** in to form **S'**



Solving the Bi-Laplacian System

$$\min_{\mathbf{v}} \mathbf{v}^T \mathbf{L}_\omega \mathbf{M}^{-1} \mathbf{L}_\omega \mathbf{v}$$

$$\text{s.t.} \quad \mathbf{v}_{H_i} = o_{H_i} \ \forall i$$

- The positions of handle vertex must be imposed as **hard constraints**

- This can be done with the row/column removal trick from last assignment:

$$A = \mathbf{L}_\omega \mathbf{M}^{-1} \mathbf{L}_\omega = \begin{bmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{bmatrix}, \quad \mathbf{b} = \mathbf{0} = \begin{bmatrix} b_f \\ b_c \end{bmatrix}$$

Instead of $A\mathbf{v} = \mathbf{b}$, solve $A_{ff}\mathbf{v}_f = \mathbf{b} - A_{fc}\mathbf{v}_c$

- (So an efficient, sparse Cholesky factorization can be used)

Pre-factoring the System

```
Eigen::SimplicialCholesky<SparseMatrixType, Eigen::RowMajor > solver;
solver.compute(A_ff); // Factorize the free part of the system
```

- solver.compute() is **slow**. After the factorization is computed, solving for different vectors is fast.
- Factorization needs to be recomputed only when the matrix A_{ff} changes (when new handles are drawn).
- For real-time performance—and full credit on the assignment—you must recompute the factorization only when necessary (not while the user is interactively transforming handles).